LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TR-444

EFFICIENT NC ALGORITHMS FOR SET COVER APPLICATIONS TO LEARNING AND GEOMETRY

Bonnie Berger
John Rompel
Peter Shor

May 1989

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

(4)

MIT/LCS/TR-444

EFFICIENT NC ALGORITHMS FOR SET COVER APPLICATIONS TO LEARNING AND GEOMETRY

Bonnie Berger
John Rompel
Peter Shor

May 1989

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR-444	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-80-C-0622
6a. NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Efficient NC Algorithms for Set Cover with Applications to Learning and Geometry

12. PERSONAL AUTHOR(S)
Berger, B. and Rompel, J.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	1989 May	14

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Set cover, computational geometry, learning theory, parallel algorithms, NC

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we give NC approximation algorithms for the unweighted and weighted set cover problems. Our algorithms use a linear number of processors and give a cover that has at most log n times the optimal size/weight, thus matching the performance of the best sequential algorithms (H, Lo, C). We apply our set cover algorithm to learning thoery, giving an NC algorithm to learn the concept class obtained by taking the closure under finite union or finite intersection of any concept class of finite VC-dimension which has an NC hypothesis finder. In addition, we give a linear-processor NC algorithm for a variant of the set cover problem first proposed by (CF), and use it to obtain NC algorithms for several problems in computational geometry.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little, Publications Coordinator	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

U.S. Government Printing Office: 1986—607-047

Unclassified

Efficient NC Algorithms for Set Cover with Applications to Learning and Geometry

Bonnie Berger*
John Rompel†

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Peter W. Shor

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract

In this paper we give NC approximation algorithms for the unweighted and weighted set cover problems. Our algorithms use a linear number of processors and give a cover that has at most $\log n$ times the optimal size/weight, thus matching the performance of the best sequential algorithms [J, Lo, C]. We apply our set cover algorithm to learning theory, giving an NC algorithm to learn the concept class obtained by taking the closure under finite union or finite intersection of any concept class of finite VC-dimension which has an NC hypothesis finder. In addition, we give a linear-processor NC algorithm for a variant of the set cover problem first proposed by [CF], and use it to obtain NC algorithms for several problems in computational geometry.

Keywords: Set cover, computational geometry, learning theory, parallel algorithms, NC

*supported in part by Air Force Grant AFOSR-86-0078

†supported by a National Science Foundation Graduate Fellowship, DARPA contract N00014-80-C-0622, and Air Force AFSOR-86-0078

1 Introduction

Given a hypergraph $H = (V, E)$, the *set cover* problem consists of finding a minimum size subset $R \subseteq V$ which covers H; i.e., for which $e \cap R \neq \emptyset$ for all $e \in E$. This is equivalent to the problem of, given a set system $\mathcal{A} \subseteq 2^X$, finding a minimum subcollection $\mathcal{A}' \subseteq \mathcal{A}$ such that $\bigcup \mathcal{A}' = X$. The set cover problem is NP-complete [K], so we will not be concerned with algorithms giving exact solutions; rather, we will consider algorithms giving approximate solutions.

The best known polynomial-time approximation algorithm for set cover is the greedy set cover algorithm [J, Lo]. Surprisingly, showing that the greedy algorithm performs well is fairly challenging, as is evident in the proofs of Johnson and Lovasz. They show that the greedy algorithm always produces a cover of size at most $\log \Delta$ times optimal, where Δ is the maximum degree of H (i.e. the maximum number of edges containing any node). However, the greedy algorithm is inherently sequential. Although RNC algorithms have been proposed which perform well for some special cases [CF, ABLP, VL], no parallel algorithm which performs well on arbitrary instances has been developed until now.

In this paper, we give a linear-processor deterministic NC algorithm that always finds a cover which is within a $\log \Delta$ factor of the optimal size. Hence, the algorithm achieves the same performance as the best sequential algorithm, both in terms of cover size and processor-time product. In addition, the algorithm works equally well for weighted hypergraphs.

We also consider a related problem, which we call the *balanced set cover* problem. Specifically, given a hypergraph $H = (V, E)$ and some fixed c, $0 < c < 1$, find a minimum size subset $R \subseteq V$ such that $|R \cap e| \geq c|e|\frac{|R|}{|V|}$ for all $e \in E$. We call this a balanced cover since for each edge, the fraction of its vertices in the cover is at least a constant times the average fraction. Clearly, balanced set cover differs from normal set cover in that instead of covering every edge once, every edge is covered multiple times in proportion to its size. Chazelle and Friedman [CF] first considered this problem and developed a polynomial-time approximation algorithm which produces an R of size $O(\frac{\log(n|E|)}{\alpha})$, where αn is the size of the smallest edge. Their algorithm is a modified version of the greedy set cover algorithm and is just as sequential in nature. In this paper, we give an NC algorithm for balanced set cover, which uses a linear number of processors, and obtains the same bound on the cover size as the sequential [CF] algorithm.

The set cover problems we consider have several applications in parallel learning theory and computational geometry. Of the two, learning is more interesting, because of all the fields in which people strive to develop parallel algorithms, learning is probably one of the most natural. In this paper, we consider a fundamental learning problem that has been solved in the sequential domain [BEHW], and solve it in parallel. In particular, we consider the problem of learning in concept classes that are formed by taking either finite unions or finite intersections of a fixed base class of finite VC dimension. We show that classes of this type are NC-learnable whenever there is an NC algorithm for finding a consistent hypothesis in the base class. The only previous parallel work on this subject is in [VL]. They give an RNC algorithm for learning s-fold unions of axis-parallel rectangles in the plane, by using a randomized set cover algorithm for specialized hypergraphs. Our general techniques solve this problem as a special case. In addition, while [VL] may produce a hypothesis with up to $s^4 \log m$ rectangles, our method will always produce one with at most $s \log m$ rectangles, due to the performance guarantee of our set cover algorithm.

In computational geometry, randomization is frequently used to construct poly-time algorithms [Cl1, Cl2, Cl3, EGS, HW, RS1, RS2]. Recently, Chazelle and Friedman [CF] showed how to make most of the randomized algorithms used in computational geometry deterministic, albeit at the cost of a substantial increase in running time. To remove randomness, they use balanced set cover. Our parallel results can be used to put many of these geometry algorithms into deterministic NC,

with the number of processors required being equal to the running time of Chazelle and Friedman's algorithm.

The remainder of this paper is divided into sections as follows. In Section 2, we present our NC algorithms for approximate set cover. In Section 3, we show how to use set cover, as well as other tools, to make the learning problem parallel. In Section 4, we outline the technique of Chazelle and Friedman and describe some of the ideas that are needed to parallelize their algorithms.

We discovered the balanced set cover results in late-February. Shortly thereafter, Motwani, Naor, and Naor [MNN] announced that they also could obtain an NC algorithm for balanced set cover. Although we have not seen their proof, they have indicated that their method uses $(\log n)$-wise independence (instead of pairwise independence — as in our method), and thus it appears to require many more processors for balanced set cover than our method. Our stronger results for general set cover and the applications to learning were discovered in mid-April, and do not seem to have been duplicated by Motwani, Naor, and Naor. Nor does it appear that these results can be naturally derived from their methods.

2 NC Algorithms for Approximate Set Cover

In this section, we consider the set cover problem: given a hypergraph $H = (V, E)$, find a minimum size subset $R \subseteq V$ which covers H, i.e. for which $e \cap R \neq \emptyset$ for all $e \in E$. As previously stated, we will consider algorithms giving approximate solutions. The best known approximation algorithm, due to Johnson and Lovasz [J,Lo], is the greedy set cover algorithm. This works by repeatedly picking the vertex of maximum degree (i.e. the one contained in the most edges), adding it to the cover, and then deleting it and all edges containing it from the hypergraph. Surprisingly, showing that this simple greedy algorithm performs well is fairly challenging, as is evident in the proofs of the following theorem.

Theorem 2.1 (J,Lo) *Let $H = (V, E)$ be a hypergraph with maximum degree Δ. Define a fractional cover of H to be a function $f : V \to \mathcal{R}$ such that $0 \leq f(v) \leq 1$ for all $v \in V$ and $\sum_{v \in e} f(v) \geq 1$ for all $e \in E$. (Note that a normal cover is a fractional one where $f(v) \in \{0, 1\}$ for all $v \in V$.) Let τ be the size of the optimal cover of H, and let τ^* be the size (i.e. $\sum_{v \in V} f(v)$) of the optimal fractional cover of H. Then the size of the cover produced by the greedy set cover algorithm is at most $(\ln \Delta + 1)\tau^* \leq (\ln \Delta + 1)\tau$.*

Remark 2.2 The following problems can easily be shown to be equivalent to set cover:

1. Given a hypergraph $H = (V, E)$, find a minimum size subset of the edges which covers V.
2. Given a bipartite graph $G = (X, Y, E)$, find a minimum size subset of X which dominates Y.

An approximation algorithm for any one of these problems can be converted to approximation algorithms for the others with the same performance guarantee.

We will also consider approximation algorithms for a variation of the set cover problem, which we call the *balanced set cover* problem: given a hypergraph $H = (V, E)$, and some fixed c, $0 < c < 1$, find a minimum size subset $R \subseteq V$ such that $|R \cap e| \geq c|e|\frac{|R|}{|V|}$ for all $e \in E$.

In this section, we will provide efficient parallel approximation algorithms for these problems. In Section 2.1, we give an NC algorithm for approximate balanced set cover, which produces an R of size $O(\frac{\log(n|E|)}{\alpha})$, where αn is the size of the smallest edge. A sequential algorithm achieving the same bound was used in [CF] to obtain deterministic algorithms for several geometry problems.

Our result allows us to parallelize these problems (see Section 4). In Section 2.2, we give an NC algorithm for approximate set cover, which achieves the same bound as the greedy algorithm (within a $(1+\epsilon)$ factor). Our algorithm builds upon the techniques used in our balanced set cover algorithm. In Section 2.3, we show how to reduce the processor count of both these algorithms to be linear. Finally, in Section 2.4, we show how to extend our set cover algorithm to handle weighted hypergraphs [C].

2.1 Balanced Set Cover

Here we consider the balanced set cover problem. Specifically, given a hypergraph $H = (V, E)$ with minimum edge size αn, we want to find a subset $R \subseteq V$, $|R| \leq 8\log(nm)/\alpha$, such that for all $e \in E$, $|e \cap R| \geq c|e|\frac{|R|}{n}$ for some constant $c > 0$.

The first way one might consider finding such an R is to simply choose R at random. It can be easily verified that if each vertex is given an independent $\frac{4\log(nm)}{\alpha n}$ probability of being included in R, then with high probability R will be good.

Another method for finding a good R is the "modified greedy algorithm" of [CF], which works as follows. First break up the edges so that every edge contains between αn and $2\alpha n$ vertices. Then modify the greedy set cover algorithm by putting weights on edges. Initially each edge has weight 1. The weight of a vertex is defined as the sum of the weights of the edges which contain it. The algorithm repeatedly picks the vertex with maximum weight. Instead of removing an edge when it is covered, its weight is cut in half. Chazelle and Friedman show that running this algorithm for $4\log(nm)/\alpha$ steps produces a good R.

Suppose we want an NC algorithm to find a good R. The first approach above gives a natural RNC algorithm, but needing full independence. The second is inherently sequential. We provide a hybrid of these two algorithms which depends on only pairwise independence, and then use Luby's method [L2] to get a deterministic NC algorithm. In this case, the hard part is getting an RNC algorithm that only uses pairwise independence.

Our approach first handles two special cases: if $r \geq n$, we let $R = V$ and quit; if $\alpha > 1/\log^2 n$ (implying $r < 8d\log^3 n$), we run the "modified greedy algorithm" directly. Otherwise, we alter the "modified greedy algorithm" as follows: rather than picking one vertex at a time, we pick a large set of vertices which significantly decrease the total edge weight. This is accomplished by picking vertices with probability $\frac{\delta}{\alpha n}$, where δ is a small constant. This implies that for each edge, the expected number of vertices picked is between δ and 2δ. (Actually, we are ignoring the fact that the size of edges decreases as they are hit. However, one can ensure that all edges not hit enough times already contain at least $7\alpha n/8$ vertices; thus, the numbers here are off by at most a factor of $8/7$.) Because we are picking such a sparse set of vertices, even assuming just pairwise independence, we can show that the probability of covering a given edge at least once is at least $\delta - 2\delta^2$. So we expect to reduce the total edge weight by a $\frac{\delta - 2\delta^2}{2}$ fraction, at least. After about $4\log m/\delta$ iterations, we expect the total weight to have dropped from m to $1/m$. In this case, every edge has weight at most $1/m$, and thus has been hit at least $\log m$ times. So $|R|$ will be about $\frac{4\log m}{\delta}\frac{\delta}{\alpha}$, and $|e \cap R|$ will be at least $\log m$, which is at least $\frac{1}{8}|e|\frac{|R|}{n}$.

To make this deterministic, we need to be able to construct a set of about δ/α vertices which causes the total weight to decrease by about $\delta/2$. Since we know that picking pairwise independently with probability $\frac{\delta}{\alpha n}$ is expected to work, we could apply the techniques of [KW,L1,ABI], trying every sample point of a small pairwise independent distribution, to get a deterministic NC algorithm. Unfortunately, this entails an $O(n^2)$ blowup in the number of processors. Instead, we will apply the techniques of [L2], first obtaining a blowup of only $O(\alpha n)$, then, in Section 2.3, showing how to achieve a linear number of processors.

To apply the techniques of [L2], we must reduce the problem to finding an $X = < X_1, \ldots, X_n >$ with $BENEFIT(X) \geq E[BENEFIT(X)]$, for some $BENEFIT$ function which is a sum of terms depending on one or two random variables each. Our first cut at a $BENEFIT$ function is to try to lower bound the decrease in total weight. Consider some edge $e \in E$ with weight w_e. A good lower bound on the weight removed on edge e is $\frac{w_e}{2} \bigvee_{i \in e} X_i$; i.e., $w_e/2$ if edge e is hit and 0 otherwise. This is not a good function to work with, since to apply [L2] we must have a function which is a sum of terms depending on one or two variables each. However, we can further lower bound by the first two terms of the inclusion-exclusion expansion,

$$\frac{w_e}{2}(\sum_{i \in e} X_i - \sum_{i,j \in e} X_i X_j) = BENEFIT_e(X).$$

It is easily verified from this that $E[BENEFIT_e(X)] \geq \frac{w_e}{2}(\delta - 2\delta^2)$. So

$$E[\sum_{e \in E} BENEFIT_e(X)] \geq \frac{w_{total}}{2}(\delta - 2\delta^2).$$

Hence, we might try to apply Luby's technique to get an X such that

$$\sum_{e \in E} BENEFIT_e(X) \geq \frac{w_{total}}{2}(\delta - 2\delta^2).$$

Why isn't this enough? Because the X output may be 1 in significantly more than δ/α places. In this case, after the correct number of iterations, every edge would be hit $\log m$ times, but R would be too large. So, we must modify the $BENEFIT$ function so that this cannot occur. Let $|X| = \sum_{i \in V} X_i$. We let

$$BENEFIT(X) = \sum_{e \in E} BENEFIT_e(X) - \alpha\delta w_{total}(|X| - E[|X|])^2.$$

Note that $BENEFIT(X)$ is still a lower bound on the weight removed by picking X.

We already argued that $E[\sum_{e \in E} BENEFIT_e(X)] \geq \frac{w_{total}}{2}(\delta - 2\delta^2)$. And,

$$\begin{aligned}
E[(\sum_{i \in V} X_i - E[\sum_{i \in V} X_i])^2] &= E[(\sum_{i \in V} X_i)^2 - E[\sum_{i \in V} X_i]^2] \\
&= \sum_{i,j \in V} E[X_i]E[X_j] + \sum_{i \in V} E[X_i^2] - \sum_{i,j \in V} E[X_i]E[X_j] - \sum_{i \in V} E[X_i]^2 \\
&= \sum_{i \in V}(E[X_i^2] - E[X_i]^2) \\
&= |V|(\frac{\delta}{\alpha n} - (\frac{\delta}{\alpha n})^2) \\
&\leq \delta/\alpha.
\end{aligned}$$

So $E[BENEFIT(X)] \geq w_{total}(\frac{\delta}{2} - 2\delta^2)$.

Lemma 2.3 BENEFIT$(X) \geq 0 \Longrightarrow \frac{\delta}{\alpha} - \sqrt{\frac{1}{\alpha\delta}} \leq |X| \leq \frac{\delta}{\alpha} + \sqrt{\frac{1}{\alpha\delta}}$.

Proof: Assume $||X| - \frac{\delta}{\alpha}| > \sqrt{\frac{1}{\alpha\delta}}$. Then $(|X| - E[|X|])^2 > \frac{1}{\alpha\delta}$. So $\alpha\delta w_{total}(|X| - E[|X|])^2 > w_{total}$ and $BENEFIT(X) < 0$. □

Plugging into [L2], we get an X with $BENEFIT(X) \geq w_{total}(\frac{\delta}{2} - 2\delta^2)$. Thus we can pick a set of size at most $\frac{\delta}{\alpha} + \frac{1}{\sqrt{\alpha\delta}}$ which causes the total weight to decrease by approximately a $\delta/2$ fraction. This allows us to get an R with the desired properties.

The running time of this procedure is $O(\log m)$ times the $O(\log^3 n)$ time required for one invocation of [L2]. The number of processors is $O(\sum_{e \in E} |e|^2 + n^2) = O(\alpha n \sum_{e \in E} |e| + n^2)$. Assuming the n^2 term is insignificant, this is αn times the size of the input, $O(\sum_{e \in E} |e|)$. In Section 2.3, we show how to achieve a linear number of processors for this problem.

2.2 Set Cover within $\log \Delta$ of Optimal

Now we turn our attention to the problem of, given a hypergraph $H = (V, E)$, finding a cover of H containing $O(\tau^* \log \Delta)$ vertices, where Δ is the maximum degree of H and τ^* is the size of the optimal fractional cover of H.

In the previous section, since a random cover worked with high probability, to get an RNC algorithm using only pairwise independence, we could simply pick a sequence of random sparse subsets and use the metric from the greedy algorithm to ensure each one contributed to a good cover. Here, however, since a random cover is not likely to be good, the behavior of our algorithm must more closely follow the behavior of the greedy algorithm.

To find a near-optimal cover, we proceed as follows. Let ϵ be a constant, $0 < \epsilon \leq 1/12$. We can cover H in a series of phases. At the beginning of phase i (phases will be sequenced in decreasing order), the hypergraph induced by the algorithm thus far will have maximum degree less than $(1+\epsilon)^i$. During this phase, we will restrict our attention to the subhypergraph induced by the vertices of degree between $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$ and will only add these vertices to the cover. In this way we only add to the cover vertices that have degrees close to the maximum, thereby emulating the greedy algorithm.

In the previous section, since the edges were guaranteed to range in size between αn and $2\alpha n$ vertices, we were able to pick vertices pairwise independently with probability $\frac{\delta}{2\alpha n}$ to get a good set. Here, however, the edge sizes can vary arbitrarily, making the task of getting a single probability impossible. If we set it too low, then the small edges will not be hit quickly enough. If we set it too high, then the large edges will be hit many times; this is undesirable since the average number of edges covered by each vertex could become much smaller than the maximum degree, thus deviating from the behavior of the greedy algorithm. Our solution is to perform a sequence of subphases. At the beginning of subphase j (again sequenced in decreasing order), all edges will contain fewer than $(1+\epsilon)^j$ vertices (which were restricted above to have degrees between $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$). During this subphase, we will repeatedly pick vertices with probability $\frac{\delta}{(1+\epsilon)^j}$, where $0 < \delta \leq 1/12$. Picking vertices with this probability will allow us to cover a $\delta/2$ fraction of the edges of size at least $(1+\epsilon)^{j-1}$, but will not cause many edges to be hit more than once, since no edge is larger than $(1+\epsilon)^j$.

More precisely, a subphase consists of a series of selection steps, performed until there are no more edges of size at least $(1+\epsilon)^{j-1}$. The selection steps are of two types. If one or more vertices cover a $\delta^3/(1+\epsilon)$ fraction of these large edges, we select one such vertex. Otherwise, we run a selection procedure which produces a collection P of vertices which cover at least $|P|(1+\epsilon)^{i-1}(1 - 6\delta - 2\epsilon\delta)$ edges, including at least a $\delta/2$ fraction of the large edges. (Later we will give both randomized and deterministic methods for performing this selection step.) In either case, the selected vertex or vertices are added to the cover and deleted, the edges covered by these are deleted, and vertices which now have degree less than $(1+\epsilon)^{i-1}$ are removed from consideration for this phase.

Theorem 2.4 *The number of selection steps is* $O(\log^2 n \log m)$.

Proof: Clearly the number of subphases is $O(\log n \log m / \log^2(1+\epsilon)) = O(\log n \log m)$. Within each subphase, each selection step removes at least a $\delta^3/(1+\epsilon)$ fraction of P_j. Thus, there are $O(\log n)$ selection steps within each subphase. □

Theorem 2.5 *The number of vertices in the cover produced by the above algorithms is at most* $\frac{(1+\epsilon)^2}{1-6\delta-2\epsilon\delta}(\log\Delta)\tau^*$, *where* Δ *is the maximum degree of any vertex, and* τ^* *is the size of the minimum fractional cover.*

Proof Sketch: Consider a selection step which adds k vertices to the cover. These vertices will cover a combined total of at least $k\hat{d}\frac{1-6\delta-2\epsilon\delta}{1+\epsilon}$ edges, where $\hat{d}$ is the current maximum degree of any vertex. This is at worst a $\frac{1-6\delta-2\epsilon\delta}{1+\epsilon}$ factor lower than what the greedy algorithm could attain. So, intuitively, one would expect to do no more than a factor $\frac{1+\epsilon}{1-6\delta-2\epsilon\delta}$ worse than the greedy algorithm. In fact, the proof of Lovasz [Lo] can be modified to give the theorem. The details will appear in a later version of this paper. □

Now we give randomized and deterministic versions of the selection procedure. Let $H_i = (V_i, E_i)$ be the current hypergraph, restricted to vertices of degree between $(1+\epsilon)^{i-1}$ and $(1+\epsilon)^i$. Also, let $E_{ij} \subseteq E_i$ contain the edges of E_i which have at least $(1+\epsilon)^{j-1}$ vertices. Furthermore, we are given that no vertex in V_i covers more than a $\delta^3/(1+\epsilon)$ fraction of E_{ij}. Then we want to return a $P \subseteq V_i$ such that P covers at least $|P|(1+\epsilon)^{i-1}(1-6\delta-2\epsilon\delta)$ edges, including at least a $\delta/2$ fraction of E_{ij}. The randomized algorithm generates P by including each vertex of V_i with probability $\frac{\delta}{(1+\epsilon)^j}$, pairwise independently. If P is good, we return it, otherwise we try again.

Theorem 2.6 *With probability at least* $1/8$, *a random* P *is good.*

Proof: To show P covers close to $|P|(1+\epsilon)^i$ edges, we will show that P covers many edges and that P is not too large. In terms of our random variables, $|P| = \sum_{k\in V_i} X_k$, which we will denote by $f_1(X)$. We will want $f_1(X) \leq \frac{\delta-2\delta^2}{(1+\epsilon)^j}|V_i|$; i.e., $f_1(X) - E[f_1(X)] \leq \frac{2\delta^2}{(1+\epsilon)^j}|V_i|$. To show this is likely, we will apply Chebyshev's inequality, so let us note that the variance of $f_1(X)$ is at most $|V_i|\frac{\delta}{(1+\epsilon)^j}$. Moreover, let us observe that $|V_i| > \frac{(1+\epsilon)^j}{\delta^3}$, since no vertex in V_i covers a $\frac{\delta^3}{1+\epsilon}$ fraction of E_{ij}. Therefore we know that with probability at least 3/4,

$$(f_1(X) - E[f_1(X)])^2 \leq 4\mathrm{Var}(f_1(X)) \leq 4|V_i|\frac{\delta}{(1+\epsilon)^j} \leq 4|V_i|^2\frac{\delta^4}{(1+\epsilon)^{2j}},$$

which implies

$$|f_1(X) - E[f_1(X)]| \leq 2|V_i|\frac{\delta^2}{(1+\epsilon)^j}.$$

A lower bound on the number of edges covered by P is

$$\sum_{e\in E_i}(\sum_{k\in e} X_k - \sum_{k,l\in e} X_k X_l) = \sum_{v\in V_i} d(v)X_k - \sum_{e\in E_i}\sum_{k,l\in e} X_k X_l.$$

The first term is bounded below by $|P|(1+\epsilon)^{i-1}$, which is at least $\frac{\delta-2\delta^2}{(1+\epsilon)^j}|V_i|$, if $(f_1(X)-E[f_1(X)])^2 \leq 4\mathrm{Var}(f_1(X))$ as before. Denote the second term by $f_2(X)$. The expectation of f_2 is at most $(1+\epsilon)^i\frac{\delta^2}{2(1+\epsilon)^j}|V_i|$. Thus with probability 3/4,

$$f_2(X) \leq 4E[f_2(X)] \leq (1+\epsilon)^i\frac{2\delta^2}{(1+\epsilon)^j}|V_i|.$$

Thus with probability 1/2, both events hold, so $|P| \leq \frac{\delta+2\delta^2}{(1+\epsilon)^j}|V_i|$ and at least

$$\begin{aligned}(1+\epsilon)^{i-1}|V_i|\frac{\delta}{(1+\epsilon)^j}(1-4\delta-2\delta\epsilon) &\geq |P|(1+\epsilon)^{i-1}\frac{\delta-4\delta^2-2\delta^2\epsilon}{\delta-2\delta^2} \\ &\geq |P|(1+\epsilon)^{i-1}(1-6\delta-2\delta\epsilon)\end{aligned}$$

edges are covered.

Now we consider the edges in E_{ij}. Similar to before, we can lower bound the number of heavy edges covered by P with

$$\sum_{e\in E_i}(\sum_{k\in e}X_k - \sum_{k,l\in e}X_kX_l) = \sum_{e\in E_{ij}}\sum_{k\in e}X_k - \sum_{e\in E_{ij}}\sum_{k,l\in e}X_kX_l.$$

We will use $f_3(X)$ to denote the first term and $f_4(X)$ to denote the second. Also, similar to before, we will show that, with high probability, $f_3(X)$ is large and $f_4(X)$ small. To bound $f_3(X)$, we will use Chebyshev's inequality. To compute $\mathrm{Var}(f_3(X))$, we rewrite $f_3(X)$ as $\sum_{k\in V_i} d_kX_k$, where d_k is the degree of vertex k in subhypergraph $H_{ij} = (V_i, E_{ij})$. Thus,

$$\mathrm{Var}(f_3(X)) = \sum_{k\in V_i} d_k^2\mathrm{Var}(X_k) \leq \frac{\delta}{(1+\epsilon)^j}\sum_{k\in V_i} d_k^2.$$

Since no vertex covers a $\frac{\delta^3}{1+\epsilon}$ fraction of E_{ij}, $d_k \leq \delta^3|E_{ij}|$ for all $k \in V_i$. Thus,

$$\begin{aligned}\mathrm{Var}(f_3(X)) &\leq \frac{\delta}{(1+\epsilon)^j}\delta^3|E_{ij}|\sum_{k\in V_i} d_k \\ &\leq \frac{\delta^4|E_{ij}|}{(1+\epsilon)^j}|E_{ij}|(1+\epsilon)^j \\ &= \delta^4|E_{ij}|^2.\end{aligned}$$

Also, $E[f_3(X)] \geq |E_{ij}|\frac{\delta}{1+\epsilon}$. Therefore, with probability 7/8,

$$(f_1(X) - E[f_1(X)])^2 \leq 8\mathrm{Var}(f_3(X)) \leq 8\delta^4|E_{ij}|^2$$

which implies $f_3(X) \geq |E_{ij}|(\frac{\delta}{1+\epsilon} - \sqrt{8}\delta^2)$. The expected value of $f_4(X)$ is at most $|E_{ij}|\frac{\delta^2}{2}$. So with probability 3/4,

$$f_4(X) \leq 4E[f_4(X)] \leq |E_{ij}|2\delta^2.$$

With probability 5/8 both conditions hold, and thus the number of edges of E_{ij} covered is at least

$$|E_{ij}|(\frac{\delta}{1+\epsilon} - \sqrt{8}\delta^2 - 2\delta^2) \geq |E_{ij}|\delta(1-\epsilon-5\delta) \geq \frac{\delta}{2}|E_{ij}|.$$

With probability at least 1/8, all four conditions hold simultaneously, and P is good. □

Theorem 2.7 *There is an NC algorithm for the selection procedure.*

Proof: Observe that in the proof above we actually showed that the following four conditions imply that P is good.

$$\begin{aligned}(f_1(X) - E[f_1(X)])^2 &\leq 4\mathrm{Var}(f_1(X)), \\ f_2(X) &\leq 4E[f_2(X)], \\ (f_3(X) - E[f_3(X)])^2 &\leq 8\mathrm{Var}(f_3(X)), \text{ and} \\ f_4(X) &\leq 4E[f_4(X)].\end{aligned}$$

To make our selection procedure deterministic, we will capture these four conditions in a *BENEFIT* function, and then apply Luby's method. Let

$$BENEFIT(X) = 1 - \frac{(f_1(X) - E[f_1(X)])^2}{4\mathrm{Var}(f_1(X))} - \frac{(f_2(X)}{4E[f_2(X)]} - \frac{(f_3(X) - E[f_3(X)])^2}{8\mathrm{Var}(f_3(X))} - \frac{(f_4(X)}{4E[f_4(X)]}.$$

Clearly, $E[BENEFIT(X)] = 1/8$. It is also clear that if $BENEFIT(X) \geq 0$, then the four conditions above are satisfied. Therefore we can apply [L2] to get a good P. □

The number of processors for our deterministic selection procedure will be $O(\sum_{e \in E_i} |e|^2 + |V_i|^2)$, which is at most n times the input size of $O(\sum_{e \in E} |e| + n)$. In the next section, we show how to obtain the same result using only a linear number of processors.

2.3 Achieving a Linear Number of Processors

In Sections 2.1 and 2.2, we presented RNC algorithms for set cover, which depend on only pairwise independence. These algorithms used only a linear number of processors. However, applying Luby's method to determinize these algorithms caused an increase in the number of processors, since we require one processor for each term of the *BENEFIT* function, expanded as a sum of functions depending on one or two variables each. The reason the *BENEFIT* functions have too many terms is that they include sums of all pairs of a subset of the random variables. To achieve a linear number of processors, we modify Luby's technique: instead of computing conditional expectations on the terms of the expanded *BENEFIT* function, we compute conditional expectations on terms of the form $\sum_{i,j \in S} X_i X_j$ directly. (Note that we can rewrite terms of the form $(\sum_{i \in S} X_i - E[\sum_{i \in S} X_i])^2$ as twice the sum of all pairs of S plus $O(|S|)$ other one-variable terms.) This is similar to the technique used by Luby to achieve a linear processor MIS algorithm [L2,L3].

We will demonstrate how to compute the conditional expectations in the simple case where the X_i's are unbiased. This easily generalizes to the case where the X_i's are identically distributed but biased. To understand this, we must review Luby's method. To find an X with $BENEFIT(X) \geq E[BENEFIT(X)]$, Luby lets $X_i = \sum_{j=1}^{l} i_j \omega_j + \omega_{l+1} \bmod 2$, where $i_1 i_2 \cdots i_l$ is the binary expansion of i. He then sets one bit of ω at a time in such a way as to maximize the conditional expectation of $BENEFIT(X)$.

We provide a way to compute

$$E[\sum_{i,j \in S} X_i X_j \mid \omega_1 = s_1, \omega_2 = s_2, \ldots, \omega_t = s_t].$$

If $t = l+1$, then we know all the X_i's, and $\sum_{i,j \in S} X_i X_j = \binom{\sum_{i \in S} X_i}{2}$. Otherwise, we partition S into sets $S_\alpha = \{i \in S \mid i_{t+1} \cdots i_l = \alpha\}$. We further partition S_α into $S_{\alpha,0} = \{i = S_\alpha \mid \sum_{j=1}^{t} i_j \omega_j \bmod 2 = 0\}$ and $S_{\alpha,1} = S_\alpha - S_{\alpha,0}$. Note that given $\omega_1 = s_1, \ldots, \omega_t = s_t$,

1. $Pr[X_i = 0] = Pr[X_i = 1] = 1/2$,
2. if $i \in S_{\alpha,j}$, and $i' \in S_{\alpha,j'}$, then $X_i = X_{i'}$ iff $j = j'$, and
3. if $i \in S_\alpha$ and $i' \in S_{\alpha'}$, where $\alpha \neq \alpha'$, then $Pr[X_i = X_{i'}] = Pr[X_i \neq X_{i'}] = 1/2$.

Therefore, conditioned on $\omega_1 = s_1, \ldots, \omega_t = s_t$,

$$E[\sum_{i,j \in S} X_i X_j] = E[\sum_{\alpha} \sum_{i,j \in S_\alpha} X_i X_j + \sum_{\alpha,\alpha'} \sum_{i \in S_\alpha} \sum_{j \in S_{\alpha'}} X_i X_j]$$

$$
\begin{aligned}
&= \sum_{\alpha} E[\sum_{i,j \in S_{\alpha,0}} X_i X_j + \sum_{i,j \in S_{\alpha,1}} X_i X_j + \sum_{i \in S_{\alpha,0}} \sum_{j \in S_{\alpha,1}} X_i X_j] + \sum_{\alpha,\alpha'} \frac{1}{4} |S_\alpha||S_{\alpha'}| \\
&= \sum_{\alpha} \left[\frac{1}{2} \binom{|S_{\alpha,0}|}{2} + \frac{1}{2} \binom{|S_{\alpha,1}|}{2} + 0 |S_{\alpha,0}||S_{\alpha,1}| \right] + \frac{1}{2} \sum_{\alpha} \frac{1}{4} |S_\alpha| \sum_{\alpha' \neq \alpha} |S_{\alpha'}| \\
&= \frac{1}{2} \sum_{\alpha} \left[\binom{|S_{\alpha,0}|}{2} + \binom{|S_{\alpha,1}|}{2} \right] + \frac{1}{8} \sum_{\alpha} |S_\alpha|(|S| - |S_\alpha|).
\end{aligned}
$$

Since there are at most $|S|$ non-empty S_α's, we can compute this using $O(|S|)$ processors.

2.4 Weighted Set Cover

In this section, we consider the problem of weighted set cover: given a hypergraph $H = (V, E)$ and a weight function $w : V \to \mathcal{R}$, and letting $w(S) = \sum_{v \in S} w(v)$, find a subset $R \subseteq V$ which covers H such that $w(R)$ is small. Note that if $w(v) = 1$ for all $v \in V$, this becomes normal set cover. Let τ^* be the minimum weight of any fractional cover of H. Then,

Theorem 2.8 (C) *There is a polynomial time algorithm which outputs a cover R of H with $w(R) \leq \tau^* \log \Delta$, where Δ is the maximum degree of H.*

Chvatal's algorithm for weighted set cover is simply the greedy set cover algorithm of Johnson and Lovasz modified to always pick the vertex with minimum cost per edge covered, i.e. minimum $\frac{w(v)}{d(v)}$. We can combine this technique with our algorithm of Section 2.2 to get

Theorem 2.9 *If $\max_{v \in V} w(v) \leq 2^{\log^c n} \min_{v \in V} w(v)$, then there is an NC algorithm, using a linear number of processors, which outputs a cover R of H with $w(R) \leq \tau^* \log \Delta$.*

Proof Sketch: Modify the algorithm of Section 2.2 so that a phase considers only those vertices with $\frac{w(v)}{d(v)}$ within $(1 + \epsilon)$ of the minimum value still left. The restriction of the weight function ensures that there will be only poly-log many phases. The structure of the subphases is as before. A slightly more complicated *BENEFIT* function is required, since the vertices can have widely varying degrees, but it is of the same basic form. The proof of Chvatal can be modified to show that this algorithm performs near-optimally as stated. □

3 Application to Learning Theory

In this section, we apply the set cover algorithm of Section 2.2 to parallel learning, a field first explored by Vitter and Lin [VL]. In particular, we provide an NC algorithm for learning in concept classes that are formed by taking either finite unions or finite intersections of a fixed base class of finite VC dimension. For example, convex polygons are defined by finite intersections of half-planes. We show that classes of this type are NC-learnable whenever there is an NC algorithm for finding a consistent hypothesis in the base class (Theorem 3.8). In obtaining this result, we employ our parallel set cover algorithm to obtain a sufficiently simple explanation of the sample data. Blumer, Ehrenfeucht, Haussler, and Warmuth [BEHW] had previously solved this problem in a polynomial-time model.

The only previous parallel work on this subject is in [VL]. They give an RNC algorithm for learning s-fold unions of axis-parallel rectangles in the plane, by using a randomized set cover algorithm which is heavily tied to their specific problem. Our general techniques apply directly to this problem. In addition, while [VL] may produce a hypothesis with up to $s^4 \log m$ rectangles, our

method will always produce one with at most $s \log m$ rectangles, due to the performance guarantee of our set cover algorithm.

The following definitions are adapted from [BEHW, VL]:

Definition 3.1 Fix a domain X. A *concept class* is a nonempty set $C \subseteq 2^X$ of *concepts*. In this paper, it is assumed that X is a fixed set, either finite, countably infinite, $[0,1]^n$, or E^n for some $n \geq 1$.

Definition 3.2 Given a nonempty concept class $C \subseteq 2^X$ and a set of points $S \subseteq X$, $\Pi_C(S)$ denotes the set of all subsets of S that can be obtained by intersecting S with a concept in C; i.e., $\Pi_C(S) = \{S \cap c \mid c \in C\}$. If $\Pi_C(S) = 2^S$, then we say that S is *shattered* by C. The *Vapnik-Chervonenkis (VC) dimension* of C is the cardinality of the largest finite set of points $S \subseteq X$ that is shattered by C. If arbitrarily large finite sets are shattered, the VC dimension of C is infinite.

Definition 3.3 For any integer $m \geq 0$, $\Pi_C(m) = \max(|\Pi_C(S)|)$ over all $S \subseteq X$ of cardinality m.

Using this notation, the VC dimension of C can be defined as the largest integer d such that $\Pi_C(d) = 2^d$, or infinity.

Definition 3.4 The length of concept c, denoted $|c|$, is the number of bits required to write c in some standard encoding.

Definition 3.5 Let C be defined as above. We say that C is *NC-learnable* if there exists an NC algorithm A that takes as input a sample of a concept in C, outputs a hypothesis in C, and has the property that for all $0 < \epsilon, \delta < 1$, and $s \geq 1$ there exists a sample size $m(\epsilon, \delta, s)$, polynomial in $1/\epsilon$, $1/\delta$, and s, such that for all target concepts $c \in C$ with $|c| \leq s$, and all probability distributions P on X, given a random sample of c of size $m(\epsilon, \delta, s)$ drawn independently according to P, A produces, with probability at least $1 - \delta$, a hypothesis $h \in C$ that has error at most ϵ such that $P[c \oplus h]$, where $\oplus$ denotes the symmetric difference.

Definition 3.6 Let $C \subseteq X$ be a concept class. By $U(C)$ we denote the closure of C under finite unions, i.e.,

$$U(C) = \{c_1 \cup \cdots \cup c_s \mid s \geq 1 \text{ and } c_i \in C,\ 1 \leq i \leq s\}.$$

Similarly, $I(C)$ denotes the closure of C under finite intersections.

Definition 3.7 Let C be a concept class. An *NC hypothesis finder* for C is an NC algorithm that, given a sample of a target concept C, returns a hypothesis in C that is consistent with the sample. Note that we do not consider randomized hypothesis finders here. The *consistency problem for* C is the problem of determining if there is a concept in C that is consistent with a given sample over X.

The existence of an NC hypothesis finder for C implies that the consistency problem for C is in NC.

Theorem 3.8 *Let C be a concept class with VC dimension $d < \infty$ such that there exists an NC hypothesis finder for C. Then $U(C)$ (resp. $I(C)$) is NC-learnable.*

We will give the proof below. First we handle some preliminaries.

Definition 3.9 Let C be a concept class defined on domain X. Let A be an NC algorithm that, given a sample of a concept in C, produces a consistent hypothesis in C. For every $s, m \geq 1$, let $S_{C,s,m}$ denote the set of all m-samples of concepts $c \in C$ such that $|c| \leq s$. Let $C^A_{s,m} \subseteq C$ denote the A-image of $S_{C,s,m}$, i.e., the set of all hypothesis produced by A when A is given as input an m-sample of a concept $c \in C$ with $|c| \leq s$. We will call $C^A_{s,m}$ the *effective hypothesis space of A for target complexity s and sample size m*. We say A is an *NC-Occam algorithm* for C if there exists a polynomial $p(s)$ and a constant α, $0 \leq \alpha < 1$, such that for all $s, m \geq 1$ the VC dimension of $C^A_{s,m}$ is at most $p(s)m^\alpha$.

Theorem 3.10 (BEHW) *Let C be a concept class with a given concept complexity measure.*

1. *If there is an NC-Occam algorithm for C then C is NC-learnable.*
2. *Let A be an NC-Occam algorithm for C with effective hypothesis space $C^A_{s,m}$ for target complexity s and sample size m. Then, if the VC dimension of $C^A_{s,m}$ is at most $p(s)(\log m)^l$ for some polynomial $p(s) \geq 2$ and $l \geq 1$, then A is an NC learning algorithm for C using sample size*

$$m = \max\left[\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{2^{l+4}p(s)}{\epsilon}\left[\log\frac{8(2l+2)^{l+1}p(s)}{\epsilon}\right]^{l+1}\right].$$

Lemma 3.11 (BEHW) *Let $C \subseteq 2^X$ be a concept class of finite VC dimension $d \geq 1$. For all $s \geq 1$, let $U_s(C) = \{\bigcup_{i=1}^s c_i \mid c_i \in C,\ 1 \leq i \leq s\}$ (resp. $I_s(C) = \{\bigcap_{i=1}^s c_i \mid c_i \in C,\ 1 \leq i \leq s\}$). Then for all $s \geq 1$, the VC dimension of C_s is less than $2ds\log(3s)$.*

Proposition 3.12 (BEHW) *If the VC dimension of H is $d \geq 0$, then $\Pi_H(m) \leq m^d + 1$.*

We now use Theorem 3.10 to demonstrate the learnability of many concept classes of the form $U(C)$ and $I(C)$ for C of finite VC dimension.

Lemma 3.13 *If C has finite VC dimension $d < \infty$ and the consistency problem for C is in NC, then for any finite set $S \subseteq X$, the sets of $\Pi_C(S)$ can be listed in NC.*

Proof Assume $S = \{x_1, \ldots, x_m\}$. To produce a list L of $\Pi_C(S)$, we proceed as follows. If $m = 1$, we use the consistency algorithm for C to check if $\{x_1\}$ is consistent, and if so return $\{\emptyset, \{x_1\}\}$, otherwise return $\{\emptyset\}$. If $m > 1$, then we recurse (in parallel) to get $L_1 = \Pi_C(\{x_1, \ldots, x_{\lfloor m/2 \rfloor}\})$ and $L_2 = \Pi_C(\{x_{\lfloor m/2 \rfloor + 1}, \ldots, x_m\})$. Then in parallel for all pairs $T_1 \in L_1$, $T_2 \in L_2$, we check the consistency of $T_1 \cup T_2$, and return $\{T_1 \cup T_2 \mid T_1 \in L_1,\ T_2 \in L_2,\ T_1 \cup T_2 \text{ consistent}\}$. The depth of the recursion is $\log m$; furthermore, since by Proposition 3.12 the size of $\Pi_C(S)$ is at most $|S|^d + 1$, we run the consistency algorithm at most $O(m^{2d})$ times in parallel at each level of the recursion. Therefore, this algorithm is in NC. □

Proof of Theorem 3.8 We consider only the case $U(C)$, the other case being similar. Let S be the set of points in an m-sample of a target concept c in $U(C)$. Our strategy will be to find a hypothesis consistent with S that is formed from the union of relatively few concepts in C; i.e. not many more than s. This problem can be formulated as a set cover problem. The set to be covered is the set of positive points of S and the sets allowed in the cover are the elements of $\Pi_C(S)$ that contain only positive points.

By Lemma 3.13, we can construct $\Pi_C(S)$ in NC. Then, in parallel, we can easily compute $\mathcal{A} = \{T \in \Pi_C(S) \mid T \text{ contains only positive points of } S\}$ and $P = \{x \in S \mid x \text{ is a positive example}\}$. We can then apply the set cover algorithm of Section 2.2 to obtain a cover of size $O(s \log m)$. For

each set in the cover, we can label the other points negative and run the NC hypothesis finder for C to produce a hypothesis in C that contains only these points of the sample. Taking the union of these concepts, we obtain a hypothesis in $U(C)$. Call this algorithm A.

We have shown that A is in NC and that given any m-sample of a concept c in $U(C)$ with $|c| \leq s$, A produces a consistent hypothesis h for this sample with $|h| \leq O(s \ln m)$. Hence the effective hypothesis space $U(C^A_{s,m})$ of A for target complexity s and sample size m contains only hypotheses such that $|h| \leq O(s \ln m)$. By Lemma 3.11, the VC dimension of $U(C^A_{s,m})$ is $O(s \log(m)(\log s + \log\log m))$. Hence, A is an NC-Occam algorithm for $U(C)$ and thus by Theorem 3.10, $U(C)$ is NC-learnable. □

4 Application to Computational Geometry

Randomization is a tool which has been used extensively to construct algorithms in computational geometry [Cl1, Cl2, Cl3, EGS, HW, RS1, RS2]. Recently, Chazelle and Friedman [CF] showed how to make most of the randomized algorithms used in computational geometry deterministic, albeit at the cost of a substantial increase in running time. Our results can be used to put many of these algorithms into deterministic NC, with the number of processors required being equal to the running time of Chazelle and Friedman's algorithm. We will very briefly outline their technique and show some of the ideas needed to make the geometric applications parallel.

To remove randomization, Chazelle and Friedman [CF] use balanced set cover. The basic technique used by Chazelle and Friedman is: First, construct a hypergraph which captures the behavior of the randomized algorithm. Next, find a balanced set cover for this hypergraph. Finally, use the set cover found in the algorithm to decide which elements to use as the randomly chosen subset in the randomized version of the algorithm. For many problems, the first and third steps are easy to do in NC. The second step is simply Section 2.1 of our paper. In general, the number of processors required in steps 1 and 3 is easily seen to be less than the number required in step 2, because of the large size of the hypergraph constructed by Chazelle and Friedman.

All the algorithms discussed in [CF] use a variation of a technique which they call probabilistic divide-and-conquer. The example they give is: given n hyperplanes in d-space and a parameter r, find a simplicial cell decomposition of d-space such that each cell intersects $O(n(\log r)/r)$ hyperplanes. This decomposition, or a similar one, is used in many range search problems [Cl1, Cl2, Cl3, EGS, HW]. It is generally used to construct a data structure which consists of a $\log n$ depth tree, with each node of the tree being a simplex, and the children of a node being the simplices determined by a random sample of the hyperplanes intersecting that node. Since the tree has only $\log n$ depth, if there is an NC algorithm for constructing a level of the tree, i.e., for finding this simplicial cell decomposition of d-space such that each cell intersects few hyperplanes, then building the entire tree is in NC.

Once we have an NC balanced set cover algorithm, we still need an NC algorithm for constructing the hypergraph which is to be covered. In [CF], a *universal triangulation scheme* is used to construct this hypergraph. A *triangulation* of an arrangement of hyperplanes in d-spaces is a decomposition of the cells of that arrangement into simplices. By *universal* it is meant that there exists a canonical scheme for dividing a cell of an arrangement into simplices which depends only on that cell, and not on the whole arrangement. The very fact that their triangulation scheme is universal (which was necessary for their algorithm) means that this triangulation is easy to achieve in parallel, as it only depends on local information.

One final thing to do is, given a set of hyperplanes in d-dimensional space, construct in NC the arrangement of cells produced by these hyperplanes. In [CF] this is done by appealing to [EOS], which is an inherently sequential algorithm. One can construct the arrangement in NC by using an

alternative algorithm which has d stages. At the kth stage, we construct the arrangement induced by the hyperplanes in all the k-flats of the arrangement. We do this by using the arrangements induced by the hyperplanes in the $(k-1)$-flats, which we constructed in the previous stage. For each k-flat we will construct a graph which will give us the cells of the arrangements in the flat as connected components of the graph. For each $(k-1)$-face in a $(k-1)$-flat contained in the k-flat, we associate two vertices of the graph. One vertex will correspond to each "side" of the $(k-1)$-face. We then construct the graph by connecting a vertex corresponding to a $(k-1)$-face to vertices corresponding to $(k-1)$-faces which are in a different $(k-1)$-flat and which see the neighboring faces of a cell of the arrangement induced in the k-flat. It is not hard to see that this can be done in NC. The connected components of this graph will give the k-faces of the arrangements in the k-flats, and as we know which $(k-1)$-faces are sub-faces of a given face, this gives us the arrangement.

5 Acknowledgements

We are grateful to Tom Leighton for helpful discussions. We thank Bob Sloan for explaining to us the fundamentals of learning theory and for giving helpful comments on that material.

References

[ABI] Alon, N., L. Babai, A. Itai, "A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem", *Journal of Algorithms*, 7, pp. 567-583, 1986.

[ABLP] Awerbuch, B., A. Bar-Noy, N. Linial, D. Peleg, *Compact Distributed Data Structures for Adaptive Routing* , STOC 1989, *to appear.*

[BEHW] Blumer, A., A. Ehrenfeucht, D. Haussler, M.K. Warmuth, "Learnability and the Vapnik-Chervonenkis Dimension", *U.C. Santa Cruz Technical Report*, UCSC-CRL-87-20, Nov. 1987. Preliminary version appeared in STOC 1986.

[CF] Chazelle, B., J. Friedman, "A Deterministic View of Random Sampling and Its Use in Geometry", *Princeton Computer Science Technical Report*, CS-TR-436, Sept. 1988. Preliminary version appeared in FOCS 1988.

[C] Chvatal, V., "A Greedy Heuristic for the Set-Covering Problem", *Mathematics of Operations Research*, vol. 4, no. 3, Aug. 1979, pp. 233-235.

[Cl1] Clarkson, K.L., "A Randomized Algorithm for Closest-Point Queries," *SIAM J. Comput.*, vol. 17, 1988, pp. 830-847.

[Cl2] Clarkson, K.L.,"New Applications of Random Sampling in Computational Geometry," *Disc. Comp. Geom. 2*, 1987, pp. 195-222.

[Cl3] Clarkson, K.L.,"Applications of Random Sampling in Computational Geometry, II," *Proc. 4th Ann. ACM Symposium Comput. Geom.*, 1988, pp. 1-11.

[EGS] Edelsbrunner, H., L. Guibas, M. Sharir, "The Complexity of Many Faces in Arrangements of Lines and of Segments," *Proc. 4th Ann. ACM Symposium Comput. Geom.*, 1988, pp. 44-55.

[EOS] Edelsbrunner, H., J. O'Rourke, R. Seidel, "Constructing arrangements of lines and hyperplanes with applications," *SIAM J. Comput.*, vol. 15, 1986, pp. 341-363.

[HW] Haussler, D., E. Welzl, "Epsilon-Nets and Simplex Range Queries," *Disc. Comp. Geom. 2*, 1987, pp. 127-151.

[J] Johnson, D.S., "Approximation Algorithms for Combinatorial Problems", *J. Comput. System Sci.*, vol. 9, 1974, pp. 256-278.

[K] Karp, R.M., "Reducibility among Combinatorial Problems.", In: *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, eds. Plenum Press, New York, 1975.

[KW] Karp, R.M., A. Wigderson, "A Fast Parallel Algorithm for the Maximal Independent Set Problem", *JACM*, vol. 32, no. 4, October 1985, pp. 762-773.

[Lo] Lovasz, L., "On the Ratio of Optimal Integral and Fractional Covers", *Discrete Math.*, vol. 13, 1975, pp. 383-390.

[L1] Luby, M., "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM J. Comput.*, vol. 15, no. 4, November 1986, pp. 1036-1053.

[L2] Luby, M., *Removing Randomness in Parallel Computation Without a Processor Penalty*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 162-173.

[L3] Luby, M., personal communication.

[MNN] Motwani, R., J. Naor, M. Naor, personal communication, March 1989.

[RS1] Reif, J.H., S. Sen, *Optimal Randomized and Parallel Algorithms for Computational Geometry*, Proc. 16th Internat. Conf. Parallel Processing, St. Charles, IL, 1987. Full Version, Duke Univ. Tech. Rept., CS-88-01, 1988.

[RS2] Reif, J.H., S. Sen, *Polling: A New Randomized Sampling Technique for Comutational Geometry*, STOC 1989, *to appear.*

[VL] Vitter, J., J-H. Lin, *Learning in Parallel*, COLT 1988, pp. 106-124.

OFFICIAL DISTRIBUTION LIST

Director
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

2 copies

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

2 copies

Director, Code 2627
Naval Research Laboratory
Washington, DC 20375

6 copies

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

12 copies

National Science Foundation
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

2 copies

Dr. E.B. Royce, Code 38
Head, Research Department
Naval Weapons Center
China Lake, CA 93555

1 copy